On the Formalization and Computational Complexity of Resilience Problems for Cyber-Physical Systems

Musab A. Alturki[1,2], Tajana Ban Kirigin[3], Max Kanovich[4], Vivek Nigam[5,6], Andre Scedrov[7], and Carolyn Talcott[8]

[1] Runtime Verification Inc., Urbana, Illinois, USA
[2] KFUPM, Dhahran, Saudi Arabia musab.alturki@runtimeverification.com
[3] Faculty of Mathematics, University of Rijeka, HR bank@math.uniri.hr
[4] University College, London, UK m.kanovich@ucl.ac.uk
[5] Federal University of Paraíba, João Pessoa, Brazil
[6] Munich Research Center, Huawei, Munich, Germany vivek@ci.ufpb.br
[7] University of Pennsylvania, Philadelphia, USA scedrov@math.upenn.edu
[8] SRI International, USA clt@csl.sri.com

Abstract. Cyber-Physical Systems (CPS) are used to perform complex, safety-critical missions autonomously. Examples include applications of autonomous vehicles and drones. Given the complexity of these systems, CPS must be able to adapt to possible changes during mission execution, such as regulatory updates or changes in mission objectives. This capability is informally referred to as resilience. We formalize the intuitive notion of resilience as a formal verification property using timed multiset rewriting. An important innovation in our formalization is the distinction between rules that are under the control of the CPS and those that are not. We also study the computational complexity of resilience problems. Although undecidable in general, we show that these problems are PSPACE-complete for a class of bounded systems, more precisely, balanced systems where the rules do not affect the number of facts of the configurations and where facts are of bounded size.

Keywords: Resilience, Planning, Formal Methods, Verification, Multiset Rewriting, Computational Complexity

1 Introduction

Cyber-physical systems (CPS) are being deployed to perform complex, safety-critical tasks, often with limited or no human intervention and in disruptive or hostile environments. Autonomous vehicles [25], for example, are a topic of intense debate among researchers, industry experts, and certification bodies, primarily due to safety concerns and the unpredictability of the environment in which they operate. The same is true for autonomous applications using unmanned aerial vehicles (UAVs) [22]. A key challenge is to ensure that these systems can perform their assigned mission even when faced with changes, such as faults and unexpected changes to the mission, such as changes in goals or changes in operational constraints. This ability to adapt is often referred to as *resilience*.

The main goal of this paper is to formalize the intuitive notion of resilience as a verification problem for CPS. We start from our previous work [13, 14], in which we proposed a Timed Multiset Rewriting (MSR) framework suitable for specification and verification of CPSes. The work addressed properties without assuming changes and considered only task realization under nominal conditions, with fixed goals and fixed regulations and policies. A key challenge is the formalization of changes against which a CPS has to be resilient. This is accomplished by distinguishing between rules that are under the control of the system and rules that are not. The latter rules specify the changes in system conditions, e.g., mission objectives, to which the system may need to adapt. The main contributions of the paper to the formalization of resilience are:

1. Extension of Timed MSR to include update rules that model changes that occur during plan execution but are outside the control of the system itself, such as changes in regulations or system goals;
2. Formal definitions for resilience as verification problems for Timed MSR systems. Intuitively, a CPS is resilient to changes if it can always accomplish its missions, even if a bounded number of changes to the mission or system have occurred;
3. Study of the complexity of resilience problems. We show that for the class of balanced systems with facts of bounded size [12], the resilience problems are in PSPACE. The PSPACE hardness follows from the complexity of the planning problem [13].

We end this section with a discussion of related work. In Section 2 we motivate the study of resilience. Section 3 gives a short overview of the timed MSR used in Section 4 to specify systems and in Section 5 to define formal resilience properties. In Section 6 we investigate the complexity of verification problems. In Section 7 we conclude with a discussion of future work.

1.1 Related work

There are many informal definitions of resilience [2, 4–6, 8–10, 15, 23, 30]. In the broadest sense, resilience is "the ability of a system to adapt and respond to changes (both in the environment and internal)" [5]. NIST [24] provides a more precise definition of resilience: "The ability to anticipate, withstand, recover, and adapt to adverse conditions, stresses, attacks or compromises on systems that use or are enabled by cyber resources." The formalization of the concept of resilience proposed in this paper captures the essence of most formulations in the literature and distinguishes it from similar concepts such as robustness, recoverability, fault tolerance, reliability, etc. Robustness, for example, "is the strength, or the ability of elements, systems, and other units of analysis to withstand a given level of stress or demand without suffering degradation or loss of function" [6].

Therefore, the main difference between robustness and resilience is that robust systems do not suffer under changes in the conditions, while resilient systems may temporarily be affected, but are capable of recovering.

There have been several attempts in the literature to formally define resilience. However, these attempts tend to focus only on specific (sometimes narrow) interpretations of resilience that are relevant to the particular application domain being considered. For instance, in the context of faulty hardware or unreliable communication media, formalizations of resilience focus on formalizing the ability of the system to compute the correct values. For example, in [27], models of resilience are defined using predicate abstraction, where a program is annotated with state abstractions that over-approximate the effects of errors on computations. A similar approach is proposed in [19], where behaviors are encoded in the system states and resilience is defined as CTL/LTL properties. Again, the properties need to be specific to the system being analyzed, and are checked using explicit-state model checking. The notion of resilience these methods capture is very narrow and rigid.

In [11], resilience (and robustness) are defined formally as constrained optimization problems in the context of learning-enabled state estimation systems in the presence of an attack. The systems are modelled using a specialized form of labelled transition systems, and the resilience property is specified as the negation of a minimization objective to be achieved within a given threshold. This formalization is used to show that the complexity of the verification problem of resilience is NP-complete. The modelling approach presented, however, is specific to a class of labelled transition systems inspired by the requirements of the application domain, and it is not clear how it can be made applicable to a wider range of systems. Furthermore, the formalization of the property is rather coarse-grained. It does not allow distinguishing active attacks from changes in goals, or define execution traces that show operationally how a resilient system may lose functionality temporarily and then recover. A similar coarse-grained, optimization-based formalization of resilience against attacks, but in the context of software obfuscation, appeared earlier in [3].

In [17, 16], resilience is formally specified as pre-condition and post-condition assertions in "Resilient Contracts" (RC) , which are contracts from the contract-based design methodology whose assertions can be probabilistic and incomplete. The definition is given in the context of multi-UAV swarm control systems. Measures of deviation from the target objective of the swarm system are encoded in the system's transition system, and used to analyze recoverability at varying degrees of achievable deviations. Similar limitations to the ones explained above apply to the RC methodology.

Our interest in resilience has been renewed by a recent talk by Vardi [31] in which he emphasised that computer science needs to recognize the tradeoff between efficiency and resilience. As our simple example will illustrate, resilience is the ability of a system to bounce back, to respond to changes that affect its correct operation and goals.

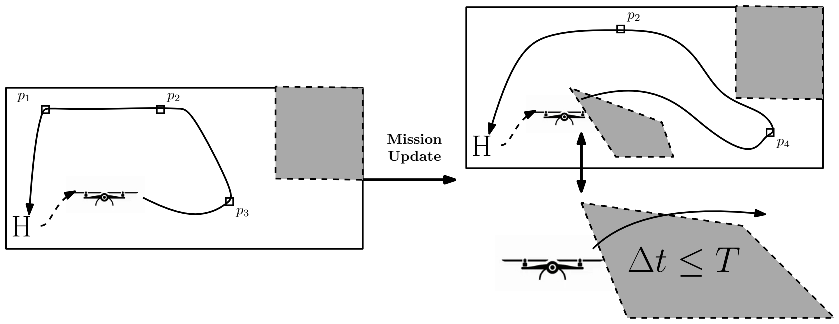

Fig. 1. Illustration of CPS resilience to mission change. Gray areas denote no-fly zones, boxes the points that the drone shall visit, and H the home base of the drone.

2 Motivating example involving drones

Resilience is actively being pursued for the development of autonomous CPS (as described in the Introduction). We illustrate the concepts used in our definitions by considering unmanned aerial vehicles (UAV), also called drones. Consider a package delivery scenario [18]. The task of the UAV is to visit a set of locations to deliver packages while complying with the policies and constraints, specifying e.g., that all points of interest should be visited within a specified time period (or deadline); that an UAV may return to home base to recharge so that it does not run out of energy; that UAV shall not fly over no-fly zones, e.g., near airports; etc. There is no particular order in which locations should be visited. However, performance quality and capability may be affected by unforeseen events, external changes, or updates that include the following: i) regulatory changes, e.g., updates to drone flight altitude restrictions; ii) policy changes, e.g., limiting energy consumption; iii) task changes, e.g., change in points to be visited; iv) deadline for accomplishing the task. A resilient drone system should be able to respond to such events by adapting and completing the task according to the new policy.

Figure 1 illustrates a mission change. A drone started with the original mission to visit points p_1, p_2, and p_3 without flying over the grey area, which is a no-fly zone. During execution, the mission is updated, as depicted in the figure on the right. The points to visit are now p_2 and p_4, but with a newly established no-fly zone. Moreover, it may be the case that the system is not robust, i.e., that it cannot be avoided that the drone flies over the new no-fly zone.

To model resilience, some requirements may be associated to updates. For example, a drone mission update will also impose that the drone has at most T time units to leave a no-fly zone. According to the informal definitions described in Section 1.1, a resilient system of drones is able to adapt to such updates and still successfully execute the mission. In this example, the drone would be able to exit the no-fly zone within T time units, visit points p_2 and p_4 and return to home base without exhausting its energy.

Different resilience requirements may demand more or less powerful CPSes, e.g., drones with larger or smaller batteries, or more or fewer drones, which will affect the overall cost of CPS. For example, if the mission change shown in Figure 1 places point p_4 too far from home base, the drone may not have enough battery capacity to complete the new mission. Similarly, if the drone is not fast enough, it may not be able to guarantee that it will leave the no-fly zone within the required time interval T. Therefore, it is important to determine during design time at which level of resilience a CPS is in relation to mission updates. A more resilient CPS will require more powerful capabilities and therefore higher costs.

The above example also illustrates the differences between resilience and reliability. Reliability (as in reliability engineering) addresses the problem of how often failures, typically in hardware, can occur and solutions to mitigate such failures, e.g., by introducing redundant hardware. In the example above, the drone system may be reliable but not resilient. For example, the drone may not be able to complete an updated mission even though there is no hardware failure, such as a motor that is not working properly. Therefore, it is not possible to directly use the rich literature on reliability to reason about the resilience of CPS.

3 Timed multiset rewriting

Assume a finite first-order typed alphabet, Σ, with variables, constants, function and predicate symbols. Terms and facts are constructed as usual (see [7]) by applying symbols of correct type (or sort). For instance, if P is a predicate of type $\tau_1 \times \tau_2 \times \cdots \times \tau_n \to o$, where o is the type for propositions, and $u_1, \ldots, u_n$ are terms of types $\tau_1, \ldots, \tau_n$, respectively, then $P(u_1, \ldots, u_n)$ is a *fact. Timestamped facts* are of the form $F@t$, where F is a fact and $t \in \mathbb{N}$ is a natural number called *timestamp*. There is a special predicate symbol $Time$ with arity zero, which will be used to represent global time. A *configuration* is a multiset of ground timestamped facts, $\mathcal{S} = \{Time@t, F_1@t_1, \ldots, F_n@t_n\}$, with a single occurrence of a $Time$ fact. Configurations are to be interpreted as states of the system. Configurations are modified by multiset rewrite rules, which can be interpreted as actions of the system. There is only one rule that modifies global time, $Tick$:

$$Time@T \longrightarrow Time@(T+1) \tag{1}$$

where T is a time variable. $Tick$ rule advances global time by one, i.e., rewrites configuration $\{Time@t, F_1@t_1, \ldots, F_n@t_n\}$ to $\{Time@(t+1), F_1@t_1, \ldots, F_n@t_n\}$. For simplicity, in this work we consider discrete time. In our previous work [12], we proposed timed MSR systems with dense time. We believe that the proposed machinery for verifying resilience also applies to dense time, but this investigation is left for future work. The remaining rules are *instantaneous* as they do not modify global time, but may modify the remaining facts of configurations (those different from $Time$). Instantaneous rules have the form:

$$Time@T, \mathcal{W}, F_1@T'_1, \ldots F_n@T'_n \mid \mathcal{C} \longrightarrow Time@T, \mathcal{W}, Q_1@(T+D_1), \ldots Q_m@(T+D_m) \tag{2}$$

where $D_1, \ldots, D_m$ are natural numbers, $\mathcal{W} = W_1@T_1, \ldots, W_n@T_n$ is a set of timestamped predicates possibly with variables, and $\mathcal{C}$ is the guard of the action which is a set of constraints involving the time variables appearing in the rule's pre-condition, i.e. the variables $T, T_1, \ldots, T_p, T'_1, \ldots, T'_n$. Constraints are of the form $T > T' \pm N$ and $T = T' \pm N$, where T and T' are time variables, and $N \in \mathbb{N}$ is a natural number. All variables in the guard of a rule appear in the rule's pre-condition. We use $T' \geq T' \pm N$ to denote the disjunction of $T > T' \pm N$ and $T = T' \pm N$. A rule $W \mid \mathcal{C} \longrightarrow W'$ can be applied on a configuration $\mathcal{S}$ if there is a ground substitution σ, such that $W\sigma \subseteq \mathcal{S}$ and $\mathcal{C}\sigma$ is true. The resulting configuration is $(\mathcal{S} \setminus W) \cup W'\sigma$. We write $\mathcal{S} \longrightarrow_r \mathcal{S}_1$ for the one-step relation where configuration $\mathcal{S}$ is rewritten to $\mathcal{S}_1$ using an instance of rule r.

A *trace* of timed MSR rules $\mathcal{A}$ starting from an initial configuration $\mathcal{S}_0$ is a sequence of configurations: $\mathcal{S}_0 \longrightarrow \mathcal{S}_1 \longrightarrow \mathcal{S}_2 \longrightarrow \cdots \longrightarrow \mathcal{S}_n$, such that for all $0 \leq i \leq n-1$, $\mathcal{S}_i \longrightarrow_{r_i} \mathcal{S}_{i+1}$ for some $r_i \in \mathcal{A}$.

Balanced systems Reachability problems for MSR systems are reduced to the existence of traces over given rules from some initial configuration to some specified configuration. Since reachability problems are undecidable in general [12], some restrictions are imposed in order to obtain decidability.[9] In particular, we use MSR systems with only *balanced* rules, i.e., rules for which the number of facts appearing in its pre-condition and in its post-condition is the same. Systems containing only balanced rules represent an important class of *balanced systems*, for which several reachability problems have been shown decidable [12]. Balanced systems are suitable, e.g., for modelling scenarios with a fixed amount of total memory. Balanced systems have the following important property [12]:

Proposition 1. *Let $\mathcal{R}$ be a set of balanced rules. Let $\mathcal{S}_0$ be a configuration with exactly m facts. Let $\mathcal{S}_0 \longrightarrow \cdots \longrightarrow \mathcal{S}_n$ be an arbitrary trace of rules $\mathcal{R}$ starting from $\mathcal{S}_0$. Then for all $0 \leq i \leq n$, $\mathcal{S}_i$ has exactly m facts. In particular, any trace without repetitions is of no more than exponential length. Moreover, the traces of exponential length may occur.*

Let $count_0$ denote exponential upper bound on the length of traces indicated in Proposition 1 stated above. In Section 6 we use the exponential upper bound $count_0$, to provide a termination for our NPSPACE procedures at least in $count_0$ steps.

Also, for some of our complexity results, we will assume an upper-bound on the size of facts, as in [12]. The size, $|F@t|$, of a timed fact $F@t$ is the total number of symbols in F, e.g., $|M(a,b,f(a,b))@t| = 6$.

[9] For a discussion on the form of rules and other conditions in the model that may affect complexity, see [12, 13].

4 Timed MSR for resilient systems

The proposed notion of resilience assumes two entities, a system and an external entity, such as the environment or regulatory authorities that mandate changes or updates to the policies that the system is supposed to comply with. We consider it "crital", i.e., unsatisfactory when the system does not adhere to such rules and guidelines. To model the two entities, we split the description of the whole scenario into a system part and a planning update part. Moreover, we consider different types of updates, including those that affect the goals of the system and those that regulate the expected behaviour of the system.

Definition 1 (Planning Configuration). *We assume a set of predicate symbols $\Sigma_P = \Sigma_G \uplus \Sigma_C \uplus \Sigma_S \uplus \{\mathsf{Time}\}$ consisting of four pairwise disjoint sets of predicates, Σ_G, Σ_C, Σ_S and $\{\mathsf{Time}\}$. Facts constructed using predicates from Σ_G are called* goal facts*, from Σ_C* critical facts*, and from Σ_S* system facts*. Facts constructed using predicates from $\Sigma_C \cup \Sigma_G$ are called* planning facts*. Configurations over Σ_P predicates are called* planning configurations*.*

For readability, we underline only planning predicates and refer to planning configurations as configurations for short.

Example 1. Predicates $\Sigma_G = \{\underline{\mathsf{Point}}, \underline{\mathsf{MinCov}}\}$, $\Sigma_C = \{\underline{\mathsf{MinBat}}, \underline{\mathsf{MinTimeToVisit}}\}$, and $\Sigma_S = \{\mathsf{Drone}, \mathsf{Visited}, \mathsf{NotVisited}, \mathsf{At}, \mathsf{BatStatus}, \mathsf{NumVisited}, \mathsf{Leq}\}$, allow the representation of information on visited points in the drone scenario with the following planning configuration:

$$\begin{aligned}\mathcal{S} = \{&\underline{\mathsf{Point}}(p(1,2))@0, \underline{\mathsf{Point}}(p(4,5))@0, \underline{\mathsf{MinBat}}(20)@0, \underline{\mathsf{MinCov}}(1)@0\} \cup \\ &\{\mathsf{Time}@4, \mathsf{Visited}(p(1,2))@2, \mathsf{At}(p(3,4))@4, \mathsf{BatStatus}(95)@4, \mathsf{NumVisited}(1)@4\}\end{aligned}$$

Remark 1. Note that the arithmetic comparisons in the MSR model are only used in time constraints, i.e., over time variables. However, we encode arithmetic conditions over non-timed variables using a binary system predicate Leq, denoting the "less or equal" relation. That is, in the (initial) planning configuration we include (persistent) facts Leq (0,0)@0, Leq (0,1)@0, Leq (1,1)@0, Leq (0,2)@0, Leq (1,2)@0, ..., for (N, M) such that $N \leq M$ up to some bound, and the $\mathsf{NLeq}(X, Y)$ facts for the remaining pairs (X, Y), for $X > Y$.The bound can be chosen to cover the numerical values of interest, such as the maximum resource values, the coordinates of the area of interest, etc.

The behaviour of the system is represented by traces of MSR rules. A system should achieve its goals while not violating certain regulations and policies, as well as restrictions related to the physical environment, such as distances and energy. This is modelled using the following concepts of goals and compliance.

Definition 2 (Critical/Goal Configurations). *A* critical configuration specification *$\mathcal{CS}$ (resp.* goal *$\mathcal{GS}$) is a set of pairs $\{\langle \mathcal{S}_1, \mathcal{C}_1 \rangle, \ldots, \langle \mathcal{S}_n, \mathcal{C}_n \rangle\}$, with each pair $\langle \mathcal{S}_j, \mathcal{C}_j \rangle$ being of the form $\langle \{F_1@T_1, \ldots, F_p@T_{p_j}\}, \mathcal{C}_j \rangle$, where $T_1, \ldots, T_{p_j}$ are time variables, $F_1, \ldots, F_{p_j}$ contains at least one critical fact (resp. goal fact),* *and $\mathcal{C}_j$ is a set of time constraints involving only variables $T_1, \ldots, T_{p_j}$. A configuration $\mathcal{S}$ is a* critical configuration *w. r. t. $\mathcal{CS}$ (resp. a* goal configuration *w.r.t. $\mathcal{GS}$) if for some $1 \leq i \leq n$, there is a grounding substitution, σ, such that $\mathcal{S}_i\sigma \subseteq \mathcal{S}$ and $\mathcal{C}_i\sigma$ evaluates to true.*

Example 2. Goal $\{\langle\{\mathsf{NumVisited}(N)@T_1, \underline{\mathsf{MinCov}}(R)@T_2, \mathsf{Leq}(R,N)@T_3\}, \emptyset\rangle\}$ denotes that the specified minimal number of points must be visited. Critical configuration specification $\{\langle\{\mathsf{BatStatus}(E)@T_1, \underline{\mathsf{MinBat}}(M)@T_2, \mathsf{Leq}(E,M)@T_3\}, \emptyset\rangle$, $\langle\{\mathsf{Time}@T, \mathsf{NotVisited}(P_1)@T_1, \underline{\mathsf{MinTimeToVisit}}(P_1, D)@T_2\ , T > T_1 + D\}\rangle\}$ denotes that the battery level should stay above the minimum allowed and that the points should be visited regularly, every D time units.

Definition 3 (Compliant Traces). *Given critical configuration specification $\mathcal{CS}$, a trace $\mathcal{T}$ is* compliant *w.r.t. $\mathcal{CS}$ if $\mathcal{T}$ does not contain any critical configuration w.r.t. $\mathcal{CS}$.*

Modelling change While system rules specify the behaviour of the system, external influences that represent changes or updates that affect the system's plan execution are modelled through update rules. All the rules used in our models are either of the form (Eq. 1) or (Eq. 2).

Definition 4 (System Rules). *A* system rule *is either the Tick rule (Eq. 1) or a rule of form (Eq. 2) such that if a planning fact is involved, then it is a permanent fact, i.e., it is not consumed by the rule.*

Definition 5 (Update Rules). *Given a planning alphabet Σ_P, a goal $\mathcal{GS}$ and a critical configuration specification $\mathcal{CS}$, an* update rule *is a rule of the form of Eq. (2) that is of one of the following type: a)* System update rule (SUR) *such that if a planning fact is involved, then it is a permanent fact; b)* Goal update rule (GUR) *that either consumes or creates at least one goal fact. If a critical fact is involved, then it is a permanent fact; c)* Critical update rule (CUR) *that either consumes or creates at least one critical fact. If a goal fact is involved, then it is a permanent fact.*

Intuitively, GUR and CUR model external influence on the system, such as regulatory changes, additional tasks, etc., while SUR model changes in the system that are not due to intentions of the system's agents, e.g., technical errors such as a drone breaking down.

Example 3. The following GUR changes the location of the points that the drone needs to visit by some given value d_G:

$$\mathsf{Time}@T, \underline{\mathsf{Point}}(X_1, X_2, X_3)@T_1, \mathsf{Visited}(X_1, X_2, X_3)@T_2, \mathsf{NumVisited}(Y+1)@T_3 \longrightarrow$$
$$\mathsf{Time}@T, \underline{\mathsf{Point}}(X_1 + d_G, X_2, X_3)@T, \mathsf{NotVisited}(X_1 + d_G, X_2, X_3)@T, \mathsf{NumVisited}(Y)@T$$

The following CUR changes the minimal time between visits by some value d_C:

$$\mathsf{Time}@T, \underline{\mathsf{MinTimeToVisit}}(P)@T_1 \longrightarrow \mathsf{Time}@T, \underline{\mathsf{MinTimeToVisit}}(P + d_C)@T$$

Definition 6 (Planning Scenario).
A planning scenario *is a tuple $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ where $\mathcal{R}$ is a set of system rules, $\mathcal{GS}$ is a goal, $\mathcal{CS}$ is a critical configuration specification, $\mathcal{E}$ is a set of update rules, and $\mathcal{S}_0$ is an initial configuration.*

Recoverability conditions We use auxiliary relations to distinguish resilience from similar properties such as robustness. Intuitively, recoverability relations specify quantitative aspects of resilience. Namely, a resilient system may not always withstand a suffered level of stress, but will recover from it in a satisfactory manner, as specified by the recoverability conditions.

Definition 7. *A* recoverability condition δ *is a binary relation over configurations. (We assume that recoverability conditions can be checked in poly-time.)*

Example 4. Recoverability conditions related to critical updates specify transitional policies before the system complies with the updated regulations and policies. For example, the time required for a system to recover from a critical situation may be bounded. After a CUR, which changes the allowed minimum battery level, the drone's energy level may be below the specified minimum. It should recharge within d time units, as specified by the following relation:

$$\{(\mathcal{S}_1, \mathcal{S}_2) \mid \mathsf{BatStatus}(B_1)@T_1 \in \mathcal{S}_1 \wedge \\ \{\underline{\mathsf{MinBat}}(M_2)@T_2, \mathsf{BatStatus}(B_2)@T_3, \mathsf{Leq}(M_2, B_2)@0\} \subseteq \mathcal{S}_2 \ \wedge \ T_3 - T_1 \leq d\}.$$

Example 5. Recoverability conditions related to goal updates specify how the new goal relates to the original goal. For example, a GUR may change the minimum number of points to visit. If the minimum coverage is increased, drones are given additional time d to complete the task, as specified by the relation:

$$\{(\mathcal{S}_1, \mathcal{S}_2) \mid \{\mathsf{Time}@T_0, \underline{\mathsf{MinCov}}(C_1)@T_1\} \subseteq \mathcal{S}_1 \ \wedge \ \mathsf{Leq}(C_1, C_2)@0 \ \wedge \\ \{\mathsf{NumVisited}(V_2)@T_2, \underline{\mathsf{MinCov}}(C_2)@T_3, \mathsf{Leq}(C_2, V_2)@0\} \subseteq \mathcal{S}_2 \ \wedge \ T_2 - T_0 \leq d\}.$$

5 Verification problems

The first problem we consider is the planning problem (or the compliance problem) which consists in checking the existence of a compliant trace showing that a system can achieve the given goal considering the given critical configuration specifications, without any updates.

Definition 8 (Compliant Planning Scenario. Planning Problem.). *A planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is* compliant *if there exists a trace τ using only $\mathcal{R}$ rules starting from $\mathcal{S}_0$ to a goal configuration w.r.t. $\mathcal{GS}$ that is compliant w.r.t. $\mathcal{CS}$. The* planning problem *consists in checking whether the given planning scenario is compliant.*

Resilience In the next verification problems, we formalize resilience under the assumption that changes do not happen too often, i.e., a resilient system should handle a bounded number of updates. There are additional inputs to the problems w.r.t. the compliance problem from Definition 8. These inputs include the number of updates allowed and recoverability conditions (Definition 7).

The resilience problems defined below are considerably more intricate than the planning problem. First, the system must be able to find a good trace, i.e., a compliant trace that reaches a goal for the given initial specification. Moreover, at any point in this trace, any one of the update rules can be applied, changing either the goal, critical configurations, or the state of the system itself. The system should be able to handle such changes, recover within the specified conditions, and find a new good trace. There may be a series of updates, and the system should be able to handle any combination of such events. Hence, rather than just finding one good trace, as in the the planning problem, the system must be able to create a set of good plans that ensures that the system can successfully adapt and reschedule after any sequence of updates.

The problems are defined recursively on the number of allowed updates.

Definition 9 **(n-Resilience w.r.t. System Updates).** *Given a natural number n, a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is* n-resilient w.r.t. system updates *if*

1. *$n = 0$, then A is compliant (see Definition 3);*
2. *$n > 0$, then there exists a compliant trace τ from $\mathcal{S}_0$ to a goal configuration $\mathcal{S}_k$ using $\mathcal{R}$ such that if for any* SUR *$r \in \mathcal{E}$ applied on some configuration $\mathcal{S}_i$ in τ, where $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, there is a compliant trace $\tau' = \mathcal{S}'_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}'_m$ using $\mathcal{R}$ such that*
   - *$\mathcal{S}'_m$ is a goal configuration;*
   - *the planning scenario $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_{i+1})$ is $n-1$-resilient w.r.t. system updates.*

Note that any update rule may be applied to any enabling configuration at any point in the trace. Following the change, a system should still be able to reach a goal. Moreover, by changing system facts, a system update should not affect compliance or otherwise the system will not be considered resilient. For example, in low temperatures battery consumption may increase, but resources should not fall to a critical level, i.e., below the minimum allowed. Similarly, if a drone malfunctions due to its electronic components being exposed to very high temperature, its performance may be degraded and the mission compromised.

The next resilience problem formalizes goal changes and involves a recoverability condition δ that relates the current goal and the new goal. A goal update changes the goals that the system must reach. Consequently, the system must provide a new trace that reaches the new goal within the conditions specified by δ, which relate the old to the new goal, and may refer to time, resources, etc. As with SUR, GUR should not compromise compliance, i.e., the newly scheduled trace should adhere at all times to the regulations and policies specified by $\mathcal{CS}$.

Definition 10 **(δ, n-Resilience w.r.t. Goal Updates).** *Given a recoverability condition δ and a natural number n, a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is* δ, n-resilient w.r.t. goal updates *if*

1. *$n = 0$, then A is compliant (see Definition 3);*
2. *$n > 0$, then there exists a compliant trace τ from $\mathcal{S}_0$ to a goal configuration $\mathcal{S}_k$ using $\mathcal{R}$ such that if for any* GUR *$r \in \mathcal{E}$ applied on any configuration $\mathcal{S}_i$ in τ, where $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, there is a compliant trace $\tau' = \mathcal{S}'_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}'_m$ using $\mathcal{R}$ such that*

– *$\mathcal{S}'_m$ is a goal configuration;*
– *$\delta(\mathcal{S}_k, \mathcal{S}'_m)$;*
– *the planning scenario $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_{i+1})$ is $\delta, n-1$-resilient w.r.t. goal updates.*

Note that a resilient system should be able to reschedule a plan at any point and, following any of the possible updates, i.e., be ready for any update at any point in its current plan. Hence, checking for the resilience of a system involves checking the existence of multiple good traces obtained by applying the different update rules from the planning scenario at different points along its current plan. For example, if some additional points of interest need to be visited, the UAV system can be given some extra time to complete the extended mission. Regardless of when the GUR occurs during the original mission execution, a resilient UAV system should be able to adapt and perform the updated task.

Resilience to critical updates takes into account the fact that the system may find itself in a critical configuration due to a CUR. Hence, after a critical update, a "grace period" allows the system to adapt. This "grace period" is specified by the recoverability condition δ and is followed by a new compliant plan that takes into account the updated critical facts.

Definition 11 **(δ, n-Resilience w.r.t. Critical Updates).**
*Given a recoverability condition δ and a natural number n, a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is δ, n-*resilient w.r.t. critical updates *if*
1. *$n = 0$, then A is compliant (see Definition 3);*
2. *$n > 0$, then there exists a compliant trace τ from $\mathcal{S}_0$ to a goal configuration using $\mathcal{R}$ such that if for any CUR $r \in \mathcal{E}$ applied on any configuration $\mathcal{S}_i$ in τ, where $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, there is a trace $\tau' = \mathcal{S}'_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}'_m \longrightarrow \cdots \longrightarrow \mathcal{S}'_{m+p}$ using $\mathcal{R}$ such that*
   – *for each j, $m \leq j \leq m+p$, $\mathcal{S}'_j$ is not critical;*
   – *$\mathcal{S}'_{m+p}$ is a goal configuration;*
   – *$\delta(\mathcal{S}_i, \mathcal{S}'_m)$;*
   – *the planning scenario $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_m)$ is $\delta, n-1$-resilient w.r.t. critical updates.*

Note that the subtrace $\mathcal{S}'_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}'_m$ may not be compliant. This distinguishes the defined property of resilience from the general notion of robustness. Resilient systems may temporarily underperform because they are severely affected by changes, but are able to adapt to updated critical specifications and continue with a compliant plan $\mathcal{S}'_m \longrightarrow \cdots \longrightarrow \mathcal{S}'_{m+p}$.

For example, if no-fly zone restrictions are updated by a CUR at a certain stage of task execution. As shown in Figure 1, it may not be possible for an UAV to avoid a newly declared no-fly zone, breaching the flight regulations. Hence, the system would reach a critical configuration. However, the recoverability conditions may specify the transition period during which the system must adapt to the new regulations. Thereafter, the resilient UAV system must comply with the new no-fly restrictions. Again, a resilient system should be able to adapt to any such update at any stage of task execution.

The most complicated verification problem involves all types of updates.

Definition 12 (δ_C, δ_G, n-Resilience). *Given recoverability conditions δ_C, δ_G and a natural number n, a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is δ_C, δ_G, n-*resilient *if*

1. *$n = 0$, then A is compliant;*
2. *$n > 0$, then there exists a compliant trace τ from $\mathcal{S}_0$ to a goal configuration $\mathcal{S}_k$ using $\mathcal{R}$, such that if for any rule $r \in \mathcal{E}$ applied on any configuration $\mathcal{S}_i$ in τ, where $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, there is a trace τ' using $\mathcal{R}$, $\tau' = \mathcal{S}'_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}'_m \longrightarrow \cdots \longrightarrow \mathcal{S}'_{m+p}$, such that*
   - *for each j, $m \leq j \leq m+p$, $\mathcal{S}'_j$ is not critical;*
   - *$\mathcal{S}'_{m+p}$ is a goal configuration;*
   - *$\delta_C(\mathcal{S}_i, \mathcal{S}'_m)$;*
   - *$\delta_G(\mathcal{S}_k, \mathcal{S}'_{m+p})$;*
   - *the planning scenario $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_m)$ is $\delta_C, \delta_G, n-1$-resilient.*

In the above resilience problems, goals and/or critical configurations may change during the trace since CUR and GUR change goal and critical facts. The system must keep pace with these updates, meet the new goals and satisfy the new requirements according to the given recoverability conditions.

Figure 1 illustrates a mission update involving both GUR and CUR, i.e., changes in the points to visit and in regulations involving no-fly zones.

Remark 2. Note that the δ_C, δ_G, n-resilience cannot be expressed directly as the combination of resilience w.r.t. system, goal and critical updates. Any combination of updates affects the original and updated missions that involve goals and critical specifications updated a multiple number of times. Note also that for $n = 0$, all the resilience problems reduce to the planning problem.

Remark 3. In problems involving critical updates, we assume that updates are not too frequent and/or that the system recovers reasonably efficiently. That is, another update does not occur until the system has recovered from the previous one. Namely, the last condition in Definitions 11 and 12 refers to resilience with a reduced number of updates, $n-1$, and the planning scenario with a new initial configuration denoting the system after the "grace period".

Resilience problems check for the existence of a "good" trace that testifies the corresponding resilience property of a given planning scenario.

Definition 13 (Resilience Problems). δ_C, δ_G, n-resilience problem *(resp. n-*resilience w.r.t. system updates*, δ_G, n-*resilience w.r.t. goal updates*, δ_C, n-resilience w.r.t. critical updates) for a given planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$, recoverability conditions δ_C, δ_G, and a natural number n, is the problem of determining whether A is δ_C, δ_G, n-*resilient *(resp. n-*resilient w.r.t. system updates*, δ_G, n-*resilient w.r.t. goal updates*, δ_C, n-resilient w.r.t. critical updates).*

6 Computational complexity results

The PSPACE lower bound for the resilience problems can be inferred from the complexity of the planning problem [13]. The aim of this section is to design non-deterministic PSPACE procedures for the resilience problems from Section 5.

For the sake of perspicuity, we confine ourselves to the resilience problem in Definition 14 below, which is the main ingredient taken from the recursive definitions in Section 5. Recall, $\delta_G(\widehat{\mathcal{S}}, \mathcal{S}'_m)$ is supposed to relate the original goal $\widehat{\mathcal{S}}$ in the main trace τ and the 'new' goal $\mathcal{S}'_m$ in the particular reaction trace τ', the result of an update action.

Definition 14. *Given a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ and a recoverability condition δ_G, let $\tau = \mathcal{S}_0 \longrightarrow \mathcal{S}_1 \longrightarrow \cdots \longrightarrow \mathcal{S}_i \longrightarrow \cdots \longrightarrow \widehat{\mathcal{S}}$ be a compliant trace leading from an initial configuration $\mathcal{S}_0$ to a goal configuration, $\widehat{\mathcal{S}}$. We say that τ is a* resilient trace *against the update rules $\mathcal{E}$ and the recoverability condition δ_G, if for each update action caused by $(r, \mathcal{S}_i)$, where an update $r \in \mathcal{E}$ is applied to a configuration $\mathcal{S}_i$ in τ, with $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, the following holds: there is a compliant 'reaction' trace $\tau' = \mathcal{S}'_{i+1} \longrightarrow \mathcal{S}'_{i+2} \longrightarrow \cdots \longrightarrow \cdots \longrightarrow \mathcal{S}'_m$, from $\mathcal{S}'_{i+1}$ to a goal configuration $\mathcal{S}'_m$ such that, in addition, $\delta_G(\widehat{\mathcal{S}}, \mathcal{S}'_m)$ is valid.*

Remark 4. Intuitively, $\delta_G(\widehat{\mathcal{S}}, \mathcal{S}'_m)$ reads that $\mathcal{S}'_m$, the new goal configuration in the particular reaction trace τ', is accepted as an adapted version of $\widehat{\mathcal{S}}$, the original goal configuration in the main trace τ.

Remark 5. According to Definition 14, given an r, we have to investigate all pairs $(r, \mathcal{S}_i)$ so that $\mathcal{S}_i$ must be available at any position inside τ. One may initially believe that we need to store the whole trace τ, which, in principle, requires *exponential size*, please see Proposition 1 in Section 3.

Remark 6. As explained in Section 3, to obtain decidability of the resilience problems, we consider balanced systems with facts of bounded size. In addition, to obtain our complexity results, we assume that recoverability conditions are recognizable in time polynomial in the size of the system, see Definition 7.

Following Remarks 5 and 6, we can easily obtain the following result:

Proposition 2. *There exists an exponential space decision procedure that determines whether, for any given planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ with a set of balanced rules $\mathcal{R}$ and an upper bound of size of facts, and a polynomial time recognizable recoverability condition δ_G, there exists a compliant trace τ leading from an initial configuration $\mathcal{S}_0$ to a goal configuration $\widehat{\mathcal{S}}$, such that τ is a resilient trace against the update rules and recoverability conditions in the sense of Definition 14.*

Resilience problems could, therefore, be reduced to compliance by generating a new compliance problem from the resilience problem. We note that while such a reduction is possible, it would result in an exponential increase of the size of the system. Notwithstanding previous points, we obtain Theorem 1, which provides a better upper bound.

Theorem 1. *There exists a PSPACE decision procedure that determines whether, for any given planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ with a set of balanced rules $\mathcal{R}$ and an upper bound of size of facts, and a polynomial time recognizable recoverability condition δ_G, there exists a compliant trace τ leading from an initial configuration $\mathcal{S}_0$ to a goal configuration $\widehat{\mathcal{S}}$, such that τ is a resilient trace against the update rules and recoverability conditions in the sense of Definition 14.*

Proof Sketch. The main idea of the proof is a dynamic execution step-by-step, not static. The following processes are run *in parallel*:

(a) The *main process*, to execute non-deterministically a main trace τ, step-by-step.

(b) For each update, $r \in \mathcal{E}$, a specific process to reschedule any branch τ' in accordance with recoverability conditions.

Recalling that NPSPACE equals PSPACE [26], we define a non-deterministic PSPACE procedure as follows: (Here, to exclude some cases, we will assume that no r is applied to the initial configuration $\mathcal{S}_0$.) By $count_0$ we denote exponential upper bound on the length of traces indicated in Proposition 1 in Section 3.

```
begin
  – $S$ is the configuration at the current step, $count$ is a counter to control
    termination, $ok$ is a Boolean to control the success.
  – Initially, $\mathcal{S} := \mathcal{S}_0$, and $count := count_0$, and $ok := true$
  – Choose non-deterministically a goal configuration, $\widehat{\mathcal{S}}$. (We assume that the
    goal configurations are recognizable in polynomial time.) The goal configu-
    ration $\widehat{\mathcal{S}}$, defined at this initial step, is intended to be the correct goal con-
    figuration appeared at the final step of our trace $\tau$ developed by induction.

  repeat   $count := count - 1$;

  – If $ok$ then, given the current $\mathcal{S}$, guess non-deterministically a non-critical
    configuration $\widetilde{\mathcal{S}}$ such that $\mathcal{S} \longrightarrow_\rho \widetilde{\mathcal{S}}$, for a regular system rule, $\rho$.
    We assume a polynomial number of system rules $\rho$, each executing in poly-
    nomial time, so we can check in polytime, if the set of such $\widetilde{\mathcal{S}}$ is empty or
    not. if this set is empty, which means that we cannot continue our trace
    (deadlock) then reset $ok := false$;  else reset $\mathcal{S} := \widetilde{\mathcal{S}}$;
    For each update, $r$, such that $r$ is applied to the $\mathcal{S}$ at hand, with $\mathcal{S} \longrightarrow_r \mathcal{S}'$,
    if $ok$ then we 'generate' the corresponding $\tau'$ as follows:
      • Here $\mathcal{H}$ stands for the configuration at the current step, $count'$ is a
        counter to control termination, $ok$ is a Boolean to control the success.
      • Initially, $\mathcal{H} := \mathcal{S}'$, and $count' := count_0$.
      • while  $ok = true$, $count' > 0$, and it is not true that
            ($\mathcal{H}$ is a goal configuration, and $\delta_G(\widehat{\mathcal{S}}, \mathcal{H})$)
        do   $count' := count' - 1$.
        If $ok$ then, given the current $\mathcal{H}$, guess non-deterministically a non-
        critical configuration $\widetilde{\mathcal{H}}$ such that $\mathcal{H} \longrightarrow_\rho \widetilde{\mathcal{H}}$, for a system rule, $\rho$.
        if such $\widetilde{\mathcal{H}}$ does not exist (deadlock) then reset $ok := false$;  else reset
        $\mathcal{H} := \widetilde{\mathcal{H}}$;
        od
```

- $ok := false$ in the case where the current $\mathcal{H}$ is not a goal configuration or $\neg\delta_G(\widehat{\mathcal{S}}, \mathcal{H})$.

until $ok = false$, or $\mathcal{S} = \widehat{\mathcal{S}}$, or $count \leq 0$.
return "success" if $ok = true$, and the current $\mathcal{S}$ is a goal configuration such that $\mathcal{S} = \widehat{\mathcal{S}}$.
end of the procedure.

Lemma 1. *There is a non-deterministic branch terminated with "success" if and only if there is a compliant trace τ leading from an initial configuration $\mathcal{S}_0$ to a goal configuration, $\widehat{\mathcal{S}}$, such that τ is a resilient trace in the sense of Definition 14.*

Bringing all together, we conclude Theorem 1. □

Remark 7. To verify that our NPSPACE procedure is correct, we play with two orthogonal paradigms in our constructions:

(a) "one r vs. exponentially many candidates $\mathcal{S}_i$ in a fixed τ";
Within Definition 14, for a fixed r we likely deal with an exponential number of $\mathcal{S}_i$, candidates for a 'good' pair $(r, \mathcal{S}_i)$ to provide a compliant trace τ' leading from $\mathcal{S}'_{i+1}$ to a goal configuration, $\mathcal{S}'_m$.
(b) "one $\mathcal{S}$ at a moment vs. polynomially many candidates r";
Within our procedure, at any moment we deal with a unique $\mathcal{S}$ and polynomial number of r's, candidates for a 'good' pair $(r, \mathcal{S})$ to initiate a compliant trace τ' leading from the corresponding $\mathcal{S}'$ to a goal configuration, $\widetilde{\mathcal{S}}$.

Compliance/reachability problem is to prove that there exists a good trace τ such that a goal P is reachable. In resilience problems we are dealing with alternating quantifiers - the problem is to prove that there exists a good trace τ such that a goal P is reachable and that for all update rules applicable to arbitrary intermediate states in τ, there exists an adapted reaction trace such that for all update rules applicable to arbitrary intermediate states on each of the adapted traces, there exists a further adapted reaction trace, etc. etc In addition to that, the algorithm has to provide, for instance, correlations between the new goals on one level and the old goals on another level. Only for a fixed number n of quantifier alternations we provide PSPACE complexity. If n is itself a part of the input, we get in fact PSPACE to the power of n.

7 Conclusions

Resilience is of great importance in today's civilization, from the Internet to logistics, finance, and environmental science, not excluding computer science. In this paper, we formalize resilience as a verification property of cyber-physical systems in a timed multiset rewriting framework. By distinguishing the rules that are under the control of CPS from those that are not, we use specific sets of traces involving changes and system recoverability to define a satisfactory system response to the new conditions. We study the complexity of resilience problems. Since the planning problem is undecidable in general [1], the resilience problems are undecidable in general. In case of systems with balanced transition rules and a bound on size of facts the PSPACE lower bound for the resilience problems follows from the PSPACE lower bound for the planning problem [13]. We note that many important cyber-physical systems are resource limited and can be naturally modelled using balanced transition rules.

In this paper, we show that the resilience problem is PSPACE-complete for the planning scenarios of Sections 4 and 5. More precisely, we show PSPACE upper bound for a version of resilience that encapsulates resilience with respect to system updates and resilience with respect to goal updates. The case of resilience with respect to critical updates is more involved because in this case we also need to allow traces that are non-compliant during the grace periods following updates. We plan to consider the complexity of this case in the future.

We also plan to consider the time bounded versions of resilience problems and their complexity for the class of Progressing Time Systems [14]. Fragments of the formal model with lower complexity of some resilience properties may be identified. Finally, we are also investigating how to automate resilience checking. The Soft Agents (SA) framework has a builtin mechanism to model environmental perturbations such as faults, weather, or obstacles [28, 29, 18, 20]. This mechanism corresponds to the use of rules not under the control of the system being considered and is thus well suited to modeling and analyzing resilience properties of cyber-physical systems such as those proposed in this paper. We plan to use SA to carry out a variety of experiments to better understand the practical aspects of checking resilience properties for different types of CPS. Some of the authors have recently proposed [21] the use of Rewriting Logic Modulo SMT for automating the generation of safety proofs for CPSes. We believe that this work can be extended so to generate resilience proofs based on the definitions proposed here. While the basic SA framework is well-suited to modeling the ability to achieve goals with acceptable outcomes, the Rewriting modulo STM approach allows us to consider recoverability issues.

We intend to study similar properties of CPSes and other complex systems, as well as compare formal definitions and computational complexities of these properties, including the realizability, survivability, recoverability, and reliability properties over infinite traces from our previous work [14]. Some of these properties could be interpreted using game theory. It would be interesting to compare our rewriting approach to the problems with the game-theoretic approach.

Acknowledgements. Ban Kirigin is supported in part by the Croatian Science Foundation under the project UIP-05-2017-9219. The work of Max Kanovich was partially supported by EPSRC Programme Grant EP/R006865/1: "Interface Reasoning for Interacting Systems (IRIS)." Scedrov was partially supported by the U. S. Office of Naval Research under award number N00014-20-1-2635. Talcott was partially supported by the U. S. Office of Naval Research under award numbers N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002.

References

1. Aires Urquiza, A., Alturki, M.A., Ban Kirigin, T., Kanovich, M., Nigam, V., Scedrov, A., Talcott, C.: Resource and timing aspects of security protocols. Journal of Computer Security **29**(3), 299–340 (2021)
2. Allenby, B., Fink, J.: Toward inherently secure and resilient societies. Science **309**(5737), 1034–1036 (2005)
3. Banescu, S., Ochoa, M., Pretschner, A.: A framework for measuring software obfuscation resilience against automated attacks. In: 2015 IEEE/ACM 1st International Workshop on Software Protection. pp. 45–51 (2015)
4. Barker, K., Ramirez-Marquez, J.E., Rocco, C.M.: Resilience-based network component importance measures. Reliability Engineering & System Safety **117**, 89–97 (2013)
5. Bloomfield, R., Fletcher, G., Khlaaf, H., Ryan, P., Kinoshita, S., Kinoshit, Y., Takeyama, M., Matsubara, Y., Popov, P., Imai, K., et al.: Towards identifying and closing gaps in assurance of autonomous road vehicles–a collection of technical notes part 1. arXiv preprint arXiv:2003.00789 (2020)
6. Bruneau, M., Chang, S.E., Eguchi, R.T., Lee, G.C., O'Rourke, T.D., Reinhorn, A.M., Shinozuka, M., Tierney, K., Wallace, W.A., Von Winterfeldt, D.: A framework to quantitatively assess and enhance the seismic resilience of communities. Earthquake spectra **19**(4), 733–752 (2003)
7. Enderton, H.B.: A mathematical introduction to logic. Academic Press (1972)
8. Henry, D., Ramirez-Marquez, J.E.: Generic metrics and quantitative approaches for system resilience as a function of time. Reliability Engineering & System Safety **99**, 114–122 (2012)
9. Holling, C.S.: Resilience and stability of ecological systems. Annual review of ecology and systematics **4**(1), 1–23 (1973)
10. Hosseini, S., Barker, K., Ramirez-Marquez, J.E.: A review of definitions and measures of system resilience. Reliability Engineering & System Safety **145**, 47–61 (2016)
11. Huang, W., Zhou, Y., Sun, Y., Banks, A., Meng, J., Sharp, J., Maskell, S., Huang, X.: Formal verification of robustness and resilience of learning-enabled state estimation systems for robotics (2020)
12. Kanovich, M., Ban Kirigin, T., Nigam, V., Scedrov, A., Talcott, C.L.: Time, computational complexity, and probability in the analysis of distance-bounding protocols. Journal of Computer Security **25**(6), 585–630 (2017)
13. Kanovich, M., Ban Kirigin, T., Nigam, V., Scedrov, A., Talcott, C.L., Perovic, R.: A rewriting framework and logic for activities subject to regulations. Mathematical Structures in Computer Science **27**(3), 332–375 (2017)
14. Kanovich, M., Kirigin, T.B., Nigam, V., Scedrov, A., Talcott, C.: On the complexity of verification of time-sensitive distributed systems. In: Protocols, Strands, and Logic, pp. 251–275. Springer International Publishing (2021)
15. Laprie, J.C.: From dependability to resilience. In: 38th IEEE/IFIP Int. Conf. On dependable systems and networks. pp. G8–G9. Citeseer (2008)
16. Madni, A.M., Erwin, D., Sievers, M.: Constructing models for systems resilience: Challenges, concepts, and formal methods. Systems **8**(1) (2020)

17. Madni, A.M., Sievers, M.: Combining formal and probabilistic modeling in resilient systems design. Procedia Computer Science **153**, 343–351 (2019), 17th Annual Conference on Systems Engineering Research (CSER)
18. Mason, I.A., Nigam, V., Talcott, C.L., Brito, A.V.D.: A framework for analyzing adaptive autonomous aerial vehicles. In: Software Engineering and Formal Methods - SEFM 2017. Lecture Notes in Computer Science, vol. 10729, pp. 406–422. Springer (2017)
19. Mouelhi, S., Laarouchi, M.E., Cancila, D., Chaouchi, H.: Predictive formal analysis of resilience in cyber-physical systems. IEEE Access **7**, 33741–33758 (2019)
20. Nigam, V., Kim, M., Mason, I., Talcott, C.: Detection and diagnosis of deviations in distributed systems of autonomous agents. Mathematical Structures in Computer Science (2022)
21. Nigam, V., Talcott, C.: Automating safety proofs about cyber-physical systems using rewriting modulo smt. In: Rewriting Logic and its Applications (WRLA) (2022)
22. NIST: Autonomy levels for unmanned systems (alfus) framework, `https://www.nist.gov/system/files/documents/el/isd/ks/NISTSP_1011_ver_1-1.pdf`
23. Pregenzer, A.: Systems resilience: a new analytical framework for nuclear nonproliferation. Albuquerque, NM: Sandia National Laboratories (2011)
24. Ross, R., Pillitteri, V., Graubart, R., Bodeau, D., McQuaid, R.: Developing cyber resilient systems: a systems security engineering approach. Tech. rep., National Institute of Standards and Technology (2019)
25. SAE: Recommended practice: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, `https://www.sae.org/standards/content/j3016_202104/`
26. Savitch, W.J.: Relationship between nondeterministic and deterministic tape classes. Journal of Computer and System Sciences **4**, 177–192 (1970)
27. Sharma, V.C., Haran, A., Rakamaric, Z., Gopalakrishnan, G.: Towards formal approaches to system resilience. In: 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing. pp. 41–50 (2013)
28. Talcott, C., Arbab, F., Yadav, M.: Soft agents: Exploring soft constraints to model robust adaptive distributed cyber-physical agent systems. In: Software, Services, and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering. LNCS, vol. 8950. Springer (2015)
29. Talcott, C., Nigam, V., Arbab, F., Kappe, T.: Formal specification and analysis of robust adaptive distributed cyber-physical systems. In: SFM 2016: Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems, LNCS, vol. 9700, p. 1–35. Springer (2016). https://doi.org/10.1007/978-3-319-34096-8_1
30. U.S. Department of Defense: Dictionary of military and associated terms, `https://fas.org/irp/doddir/dod/jp1_02.pdf`
31. Vardi, M.: Efficiency vs. resilience: What COVID-19 teaches computing. Communications of the ACM **63**(5), 9–9 (2020)